How to Develop User-Friendly Macros

Sy Truong, Meta-Xceed, Inc, Fremont, CA

Abstract

SAS macros help automate tasks that are done repeatedly. However, if they are not easy to use or debug, they may never be used. This paper will describe approaches to developing and maintaining SAS macros that are easy to use. Some of the topics covered include:

- effective documentation of macro header
- portable code for use with different OS
- error and warning message handling
- paper and online documentation
- use of nested macros and nested macro variables
- keeping macros simple for debugging

A little effort can go a long way towards creating a successful SAS macro. This paper will present tips and techniques that are not always obvious. Besides getting the resulting numbers to the user, a user friendly macro can enhance the entire experience.

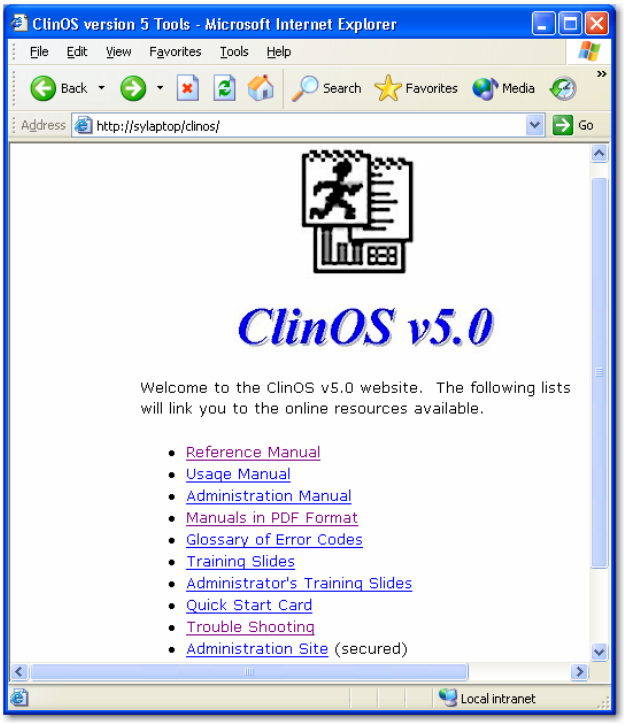
Introduction

SAS macros are great at automating repetitive tasks as a code generator. However, there are some features of macros that make them difficult to understand and debug. The syntax of SAS macros is similar to that of traditional SAS data step but there is a level of abstraction. To accomplish its code generating function, SAS macros add percent signs (%) and andpersand (&) in front of specified commands and variables. This layer can be confusing since it requires you to resolve the macro before understanding what is being processed. This confusion is compounded when things are nested. Macro variables can be nested by resolving into other macro variables. Macros themselves can call other macros which creates a nested looping structure. Macro code can sometimes be spaghetti code. It is therefore helpful for the user and the person maintaining the macro to make the macros user friendly.

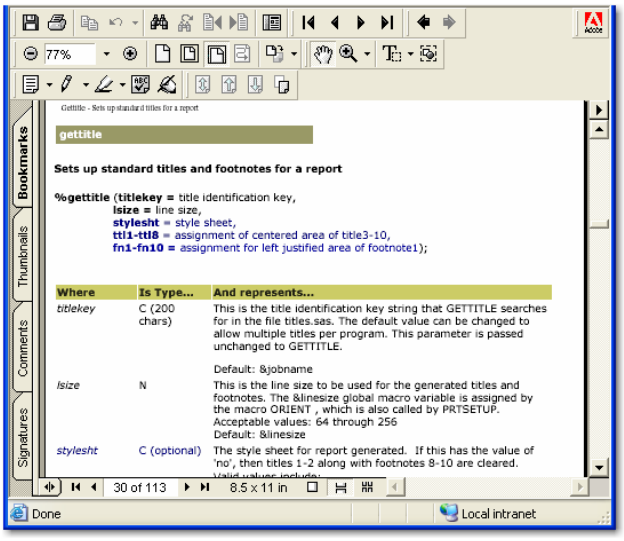
Documentation

One of the most straightforward ways to make SAS macros user friendly is to enhance the documentation that accompanies the macro. This is an essential part of software that some programmers overlook when delivering macros to users. There are many different forms of documentation and the more that is available, the greater the chance of connecting with the user. This paper will summarize eleven different types of documentation. The content can overlap among the different types but each method has its own strengths.

1. HTML – Documentation can be delivered as HTML pages within an intranet or Internet. This is readily available to all users. The website has the following advantages:
    a. Accessibility from any computer connected to the network.
    b. Hyperlinks to quickly navigate to specified content.
    c. Search engine to find specific content.
    d. Graphics for capturing screen shots and other diagrams for more effective communication.



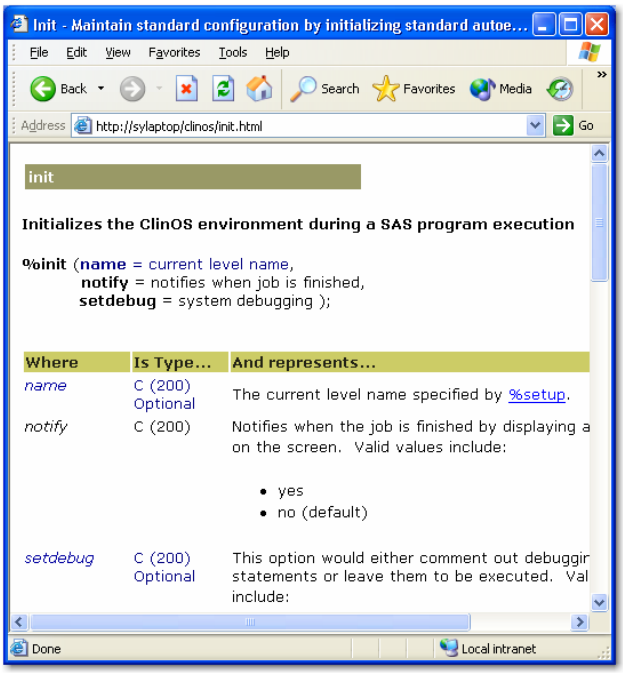
2. PDF - Similar to HTML, a PDF file can be delivered within an intranet or Internet. Its strengths are similar to HTML with some distinguishing features:
    a. Can be delivered as one file, as in an email as an attachment, in case the user does not have network access.
    b. PDF format is more consistent for printing on physical paper.
    c. The content is locked from changes.


Bookmarks
Thumbnails
Comments
Signatures
gettitle
Sets up standard titles and footnotes for a report
%gettitle (titlekey = title identification key,
lsize = line size,
stylesht = style sheet,
ttl1-ttl8 = assignment of centered area of title3-10,
fn1-fn10 = assignment for left justified area of footnote1);
Where
Is Type...
And represents...
titlekey
C (200 chars)
This is the title identification key string that GETTITLE searches for in the file titles.sas. The default value can be changed to allow multiple titles per program. This parameter is passed unchanged to GETTITLE.
Default: &jobname
lsize
N
This is the line size to be used for the generated titles and footnotes. The &linesize global macro variable is assigned by the macro ORIENT , which is also called by PRTSETUP.
Acceptable values: 64 through 256
Default: &linesize
stylesht
C (optional)
The style sheet for report generated. If this has the value of 'no', then titles 1-2 along with footnotes 8-10 are cleared.
30 of 113
8.5 x 11 in
Done
Local intranet

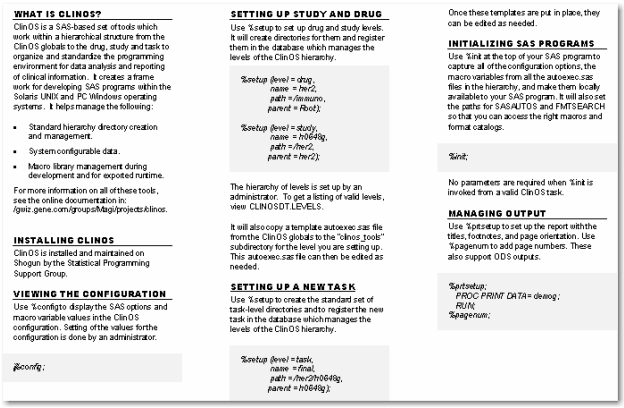
3. Reference Manual – The information is indexed in a way that a user needing to look up syntax for a particular use of a parameter can navigate to it quickly. If HTML or PDF is used, hyperlinks can speed up the cross reference among documents. Some of the

components that make up the reference material of a macro include:

a. A short one sentence description that explains what the macro does.
b. The macro name along with all its parameters. Each parameter is followed by a short description.
c. A more detailed explanation of the macro parameters including data type, valid values and description.
d. Detailed explanation of the macros noting all exceptional error conditions and default behaviors.
e. Diagrams showing how it fits into the larger scheme among other tools.
f. Example macro calls with sample parameter values.



4. Usage Manual – This is sometimes confused with the reference manual. It is different in that rather than being organized to look up specific syntax, it is organized by task. A task is what a user needs to do for a specified job. The tasks are grouped and ordered to flow in the chronology of a user's work flow. It describes the content in full English rather than bulleted abbreviation. There are links or references back to the reference manual when syntax is needed. Proper indexing and use of figures and diagrams are also very helpful.
5. Administration Manual – This can be in the format of both a reference and user manual but it is specifically geared towards power users and administrators. It is more effective to separate these topics since the audience is different. This makes the content more accessible and relevant to the specified audience.
6. Glossary of ERROR and WARNING – This contains a complete list of all the ERROR and WARNING messages generated by the macro. The messages themselves should be clear enough for the user to understand. This list will further explain the situations and details of the condition. More importantly, it suggests steps for the user to take to resolve the problem or find resources to circumvent the problem.
7. Training Material – This is most commonly power point slides. Along with slides, notes explaining the bullet points in full English can help users understand the content in the event that they are not able to attend the instructor-based training course. To keep the pace and content of the training interesting, quizzes can be utilized to test the user's acquisition of the information for each section.
8. Quick Card – This is a physical threefold double sided card stock glossy that contains the essentials for the set of macros. It has strip down content to include the very bare essentials. The card is intended to get the user up to speed quickly at the beginning of the learning curve.


WHAT IS CLINOS?
ClinOS is a SAS-based set of tools which work within a hierarchical structure from the ClinOS globals to the drug, study and task to organize and standardize the programming environment for data analysis and reporting of clinical information. It creates a frame work for developing SAS programs within the Solaris UNIX and PC Windows operating systems. It helps manage the following:
Standard hierarchy directory creation and management.
System configurable data.
Macro library management during development and for exported runtime.
For more information on all of these tools, see the online documentation in: /gwiz.gene.com/groups/Magi/projects/clinos.
INSTALLING CLINOS
ClinOS is installed and maintained on Shogun by the Statistical Programming Support Group.
VIEWING THE CONFIGURATION
Use %configto display the SAS options and macro variable values in the ClinOS configuration. Setting of the values for the configuration is done by an administrator.
%config;
SETTING UP STUDY AND DRUG
Use %setup to set up drug and study levels. It will create directories for them and register them in the database which manages the levels of the ClinOS hierarchy.
%setup (level = drug,
name = hrc2,
path = /immuno,
parent = Root);
%setup (level = study,
name = h0649g,
path = /hrc2,
parent = hrc2);
The hierarchy of levels is set up by an administrator. To get a listing of valid levels, view CLINOSDT.LEVELS.
It will also copy a template autoexec.sas file from the ClinOS globals to the "clinos_tools" subdirectory for the level you are setting up. This autoexec.sas file can then be edited as needed.
SETTING UP A NEW TASK
Use %setup to create the standard set of task-level directories and to register the new task in the database which manages the levels of the ClinOS hierarchy.
%setup (level = task,
name = final,
path = /hrc2/h0649g,
parent = h0649g);
Once these templates are put in place, they can be edited as needed.
INITIALIZING SAS PROGRAMS
Use %init at the top of your SAS program to capture all of the configuration options, the macro variables from all the autoexec.sas files in the hierarchy, and make them locally available to your SAS program. It will also set the paths for SASAUTOS and FMTSEARCH so that you can access the right macros and format catalogs.
%init;
No parameters are required when %init is invoked from a valid ClinOS task.
MANAGING OUTPUT
Use %ptsetup to set up the report with the titles, footnotes, and page orientation. Use %pagenum to add page numbers. These also support ODS output.
%ptsetup;
PROC PRINT DATA= demog;
RUN;
%pagenum;

9. FAQ or Trouble Shoot – This list contains the most common questions or problems that users run into. It can explain an ERROR condition similar to the ERROR and WARNING glossary. In addition, it also covers situations or scenarios that may happen or how the macro interacts with other data or programs. If the list is long, a search engine can be used to enhance the user's ability to find a particular problem.
10. Wish List – A list of features considered for the next release. It is most effective to put all ideas down and share them with the users. This allows the users to feel that their concerns are being met. It also allows the features to be evaluated with more time and perspective compared to an ad hoc approach to applying changes to the code.
11. Variable List – A list of all the macro variables defined in each macro is useful. This will ensure that certain common macro variables such as &i for index does not conflict with the user's use of the same macro variable.

It is not required that all of these documentation methods be used, especially for small macros. However, for larger sets of macros that form a system, it is recommended that most of these methods be implemented. Some of the content may be redundant but different users absorb information in different ways. It is therefore more effective to have many alternatives.

Coding

There are many coding conventions that you can follow to enhance the readability of your SAS macros. This section is not a comprehensive list but rather recommendations for consideration. The general rule is that whatever convention you decide to adopt, be religiously consistent throughout all of your macros.

- Standard Header Block – This is the comment section that appears at the beginning of each macro. This usually contains the name of the macro and a description of the macro along with explanation of the parameters. I recommend using "display manager abbreviations" which can play back a header template. More examples of this can be found in a paper located at: http://www.pharmasug.org/2003/BestPapers/cc025.pdf
- Named Parameters – If the macro does not use any or just one parameter, named parameters are not

necessary. However, when there are multiple parameters, it is recommended that you use named parameters since it makes the macro and the call to the macro more concise and explicit.

- Use SYSTASK – When you need to issue a command to the operating system, use SYSTASK instead of the X command. SYSTASK has more options including the ability to kill a task upon request.

```
systask command
%unquote(%str(%'&domkdir%')) wait;
```

You can type the command in directly to create a directory. In this example, this macro runs on multiple operating systems. It therefore issues different commands depending on the current operating system.

- Standard Code When Possible – What is meant by standard code is code that you may see in a data step. For example:

```
data _null_;
   if (exist('clinosdt.levels') = 0)
   then do;
      call symput('status','end');
      put "ER" "ROR: [snapshot] was
      unable to find ...";
   end;
run;
```

This is friendlier than:

```
%if ("&exist" = "&valid") %then %do;
   %put ERROR: [snapshot] was unable
   to find….
   %let status=end;
%end;
```

The code may seem shorter in macro form but the macro variables and the syntax have to be resolved and reevaluated before it can be fully understood.

- Local Macro Variables – When macros variables are created inside a macro, they are by default considered local. If possible, keep the use of macro variables local so that they will not interfere with other macros. If you have to define something that spans macros, I would consider putting it into a variable stored in a work area dataset. This way, it circumvents the problem of using conflicting global macro variables.
- Nested Macro Variables – Whenever possible, avoid using nested macro variables. That is a macro variable that resolves into another macro variable. The reference becomes &&macrovar. This makes it even more abstract for the user and can be very confusing. A strategy to avoid this is to store values in work area datasets. This way, you can use data step logic in resolving values that is clearer and easier to work with.
- Commenting Style – There are two styles for SAS comments.
    - `/*** Comment Style 1 ***/`
    - `*** Comment Style 2 ***;`

  Always use comment style 2 whenever possible. Reserve comment style 1 only when used with debugging. This allows you to block sections of code easily but they cannot be nested. If you stick to style 2, users can use style 1 to block off code if they need to.
- Explicit Reference – When referring to a work area dataset, always put in the name work.dataname. The default is if you leave out the “work.”, SAS will interpret this as the work area. However, if you explicitly spell it out, it is more consistent with other references and makes it clearer for the user.

The goal is to make the code simple yet functional. It is analogous to writing a book at the seventh grade level. If your goal is to make the information reach your audience, you would stay away from the use of esoteric jargon and prose. In a similar way, when possible, keep the code simple so a novice programmer can easily pick up the meaning of your program.

ERROR Checking

One of the keys in enhancing the users experience is for the macro to perform error checking. First, brainstorm all the problems that a user would run into, and then capture these conditions. You would then present the user with a friendly message describing the situation. This is much clearer and less confusing than the program crashing on its own. While brainstorming on all the ways a user can run into problems, consider the following conditions:

- Missing Required Parameters

```
**** Check for case where required
parameters are missing ***;
if (compress(param) = '') and
(left(trim(lowcase(print))) ne 'yes')
then do;
   put ' ';
   put 'WAR' 'NING: [config] was missing
   required PARAM specification.';
   put ' ';
   call symput('status','end');
end;
```

- Valid Values – This can sometimes be combined with checking for missing values. It is particularly useful to check for parameters with expected distinct values.

```
*** Handle invalid specifications ***;
if (setdebug ne '') and
(lowcase(setdebug) not in('yes','no'))
then do;
   put "WAR" "NING: [init] has an invalid
   SETDEBUG selection.  Valid values are
   'yes' or 'no'.";
   put "NOTE: The value of 'no' is set
   for SETDEBUG by default.";
   setdebug = 'no';
end;
```

- Stopping Criteria – After an error condition is found, it may make sense to stop the entire macro since the rest of the logic is no longer valid. From the examples above, you can see a new macro variable named status and set to the value of ‘end’. Once this condition is met, you can skip the next section.

```
%if ("&status" ne "end") %then %do;
...
```

- Valid Path – In cases where the parameters reference a physical path on disk, you can verify if the path is valid before continuing.

```
*** Verify if path exists ***;
if (fileexist(path) = 0) then do;
   put ' ';
   put 'ER' 'ROR: [logeval] path: ' path
   'does not exist.';
   put ' ';
   call symput('status','end');
end;
```

- *%nrbquote* – Whenever there is open text value for a parameter, use *%nrbquote*. This would handle unbalanced quotes or special characters.

```
%let ttl2 = %nrbquote(Summary of N (%) of
Adverse Events) ;
```

- Existence of Files – If the parameters reference files, verify if they exist before continuing. If your macro creates a file, this same technique can be used to present a NOTE message confirming that the file is created successfully.

```
if fileexist('c:\mypath\myfile')= 0
then do; …
```

- Existence of Expected Datasets – Similar to existing path or files, if there is a reference to a dataset, verify if it exists before continuing.

```
if exist('clinosdt.levels')…
```

- Case Insensitive Comparisons

```
if (lowcase(param) = 'genmax') then
do;…
```

- Dataset Availability – It is possible that the dataset exists but it is locked or corrupted. In this case, verify if it is accessible before processing it.

```
%let
dsid=%sysfunc(open(clinosdt.levitem,i));
```

- OS differences – If your macro is intended to operate on more than one operating system, you would need to check for error conditions differently depending on the current OS. Here is an example of how slashes are treated differently.

```
** Capture the proper OS slash ***;
cur_os = symget('sysscpl');
if (index(cur_os,'WIN_') > 0) then slash
= "\";
else slash = "/";
call symput('slash',slash);
```

- ERROR Checking with SAS System error variables: http://jeff-lab.queensu.ca/stat/sas/sasman/sashtml/lrcon/z1104667.htm

There are endless possibilities for error checking. This is driven by the type of data and parameters which you are using. The recommended list above is intended to spur ideas for you to be thorough with your error checking. In general, it is a good idea for all the error conditions to be checked at the very top of the macro so that no resource is wasted. If you can stop at the beginning, the log produced will be short and concise. This makes it much easier for users to debug.

ERROR Messaging

The messages that users get when something goes wrong is the main interface you have when they interact with your macros. Other types of interactive software may display dialog boxes, but in this case, text messages in the SAS log is all you have to work with. The more consistent and concise you can make your messages, the more effective they will be. The following are suggestions to effective error messaging.

- SAS Standards – Similar to SAS, precede your messages with either **ERROR:**, **WARNING:**, or **NOTE:**. The error category specifies that the macro cannot continue with the current condition. Warnings alert users to possible problems but the macro can still operate. Note messages are useful for confirmation information. Log evaluation tools and viewers use these key headers to highlight through colors. It behooves you to take advantage of this convention.

- Error Source – Following the header text of ERROR:, WARNING:, and NOTE:, it is recommended that you specify the name of your macro in square brackets. For example:

```
ERROR: [snapshot] was unable to find
system levels database.
```

In this case, the macro named "snapshot" is used. This is particularly helpful when you have several macros. It makes it easy to identify which macro the message is pertaining to and that it is not a SAS error message.

- Consistent Style – Keep the message short. If possible, make it only one sentence with one line of text. Keep the verb tense, grammar, casing and punctuation consistent and correct. One suggestion that fits the recommended style is to use the macro name in square brackets as the main subject followed by a verb and text explaining what has happened.

- Code Parsers – When users search the log for ERROR messages and if MACROGEN or SYMBOLGEN options are used, your conditional error message is captured even if the code is not executed. That is, when they search for the text "error" they will see the SAS code that generates the error message. To avoid this problem, you can break up the word ERROR or WARNING so that only when the condition is met do users see it in the log. Here are a couple of examples:

```
put "ER" "ROR: [init] was unable to find
the ...";

put "WAR" "NING: [setup] was unable to
create ...";
```

- Provide Examples – In the event that a parameter is incorrectly used, an error message is presented. It is helpful to follow the error message with an additional note giving an example of the proper usage of that parameter. For example:

```
put "ER" "ROR: [init] had an invalid
parameter ...";
put "NOTE: [init] accepts values of
'yes' and 'no' for this STARTUP
parameter.";
put %str('      For example
%init(startup=yes);');
```

Reporting

Besides generating messages in the log, another form of communication with the user is to produce output. The most common forms of output are those produced by PROCs and output datasets. This will be generally referred to as "reporting"

output. The following are some general guidelines that will help make your macros more accessible to users.

- Keep it Simple – There are many fancy ODS and PROCs to help make your report look fancy. When formatting is important, do take advantage of these features. However, if all you need to do is to deliver some status information, keep it simple with a PROC PRINT or perhaps a PROC REPORT. By default, this is delivered as plain text to the output window or a .LST file.
- Create Data in Work – No matter how fancy your reporting is, there will be some custom special requests. Rather than trying to make the report meet everyone's needs, a SAS dataset can be created to accompany your report. This dataset contains the same information presented in your report. It is one PROC away from your report. Keep this dataset in the SAS work area and include information about its attributes in the documentation. This way, users who want to generate their own customized reports can use their own PROC upon this dataset.
- User Name and Date Time – The work area dataset created contains content pertinent to the information that is delivered to the users. In addition to the content, it is recommended that two additional variables are added. The administrative information can be beneficial in understanding who and how the macros are used.
    - User Name – The current user that is running this macro.
    - Date Time – The current date and time.

```
usrname = symget('SYSUSERID');
datetime = datetime();
```

- Attributes – Whenever you define a work area dataset or a permanent dataset, it is recommended that you assign descriptive attributes. A dataset for example also contains a dataset label. SAS variables can contain formats and other attributes.

```
data work.report (label="Sample Report
   for Macro");
   attrib program length=$80 label="SAS
   Program";
   attrib macro length=$80  label="SAS
   Macro Name";
   attrib datetime length=8
   label="Current Date Time"
   format=datetime13.;
run;
```

Conclusion

There are many fascinating developments with SAS in version 9 and future releases. This gives users more options and tools to effectively analyze and report their data. SAS macro language, however, has not changed a whole lot, but it remains a viable tool. It gives SAS programmers a way of encapsulating a commonly used task into a module which can be used repeatedly. It acts as a code generator that helps users become more efficient. Even though SAS macros have not changed, they still have many pertinent features and options. If used carelessly, the confusion created from the spaghetti code will waste time than it saves. It is therefore useful to create macros that are simple yet effective. Since the macros may be used by users other than the original programmer, it is important to be thorough and complete with the accompanying documentation. By following some guidelines and being consistent with coding and error messaging, macros can be a very powerful and useful tool.

References

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

About the Author

Sy Truong is a Systems Developer for Meta-Xceed, Inc. They may be contacted at:

Sy Truong
48501 Warm Springs Blvd. Ste 117
Fremont, CA 94539
(510) 713-1686
sy.truong@meta-x.com